Towards Parallel Access of Multi-dimensional, Multi-resolution Scientific Data

Sidharth Kumar, Valerio Pascucci
SCI Institute, University of Utah
Salt Lake City, Utah

Venkatram Vishwanath, Philip Carns, Robert Latham, Tom Peterka, Michael Papka, Robert Ross
Argonne National Laboratory
Argonne, Illinois

Abstract— **Large scale scientific simulations routinely produce data of increasing resolution. Analyzing this data is key to scientific discovery. A critical bottleneck facing the analysis is the I/O time to access the data. One method of addressing this problem is to reorganize the data in a manner that simplifies analysis and visualization. The IDX file format is an example of this approach. It orders data points so that they can be accessed at multiple resolution levels with favorable spatial locality and caching properties. IDX has been used successfully in fields such as digital photography and visualization of large scientific data, and is a promising approach for analysis of HPC data. Unfortunately, the existing tools for writing data in this format only provide a serial interface. HPC applications must therefore either write all data from a single process or convert existing data as a post-processing step, in either case failing to utilize available parallel I/O resources.**

In this work, we provide an overview of the IDX file format and the existing ViSUS library that provides serial access to IDX data. We investigate methods for writing IDX data in parallel and demonstrate that it is possible for HPC applications to write data directly into IDX format with scalable performance. Our preliminary results demonstrate 60% of the peak I/O throughput when reorganizing and writing the data from 512 processes on an IBM BG/P system. We also analyze the remaining bottlenecks and propose future work towards a more flexible and efficient implementation.

Keywords-Parallel IO, Multi dimensional data;

I. INTRODUCTION

The increase in computational power of supercomputers is enabling unprecedented opportunities to advance science in numerous fields such as climate science, astrophysics, cosmology and material science. These simulations routinely produce larger quantities of raw data. A key requirement is to analyze this data and transform it into useful insight. A critical bottleneck being faced by analysis applications is the I/O time to read and write data to storage.

IDX provides efficient, cache oblivious, and progressive access to large-scale scientific data by storing the data in a hierarchical Z order [1]. It makes it possible for scientists to interactively analyze and visualize data of the order of several terabytes [2]. IDX has been used successfully in fields such as digital photography [3] and visualization of large scientific data [2] and is promising for analysis of HPC data as well [4]. ViSUS, an IDX API, is serial in nature which limits the use of IDX to relatively small scale datasets. To overcome this problem, we have developed a parallel API to transform large-scale scientific data to IDX format. It utilizes the computation resources of each compute node to efficiently calculate the HZ ordering. It then coordinates file system access using collective communication to write the data set in parallel.

Development of the parallel IDX API is the culmination of observations and experiments made over different versions of a prototype API. We began by evaluating the use of the existing ViSUS library in a parallel environment. We then constructed a prototype API, called PIDX, that allows data to be written in parallel. By analyzing this prototype, we were able to identify and address inefficiencies in both the I/O strategy and data order computation. The prototype API demonstrates that it is possible for HPC applications to write data directly into IDX format with scalable performance. We will use this prototype as a platform for future work in developing a more flexible and efficient implementation.

The remainder of this paper is organized as follows: We present relevant background information on the IDX Data format in Section 2 and describe ViSUS, a serial IDX API, in Section 3. We present our work on writing IDX data in parallel in Section 4 and discuss performance optimization next. We evaluate the performance of our parallel IDX prototype in Section 6 and finally conclude and discuss our plans for further research.

II. IDX DATA FORMAT

IDX enables fast and efficient access to large scale scientific data. In IDX, data is organized into multiple levels of resolution, making it easy to query data of any desired size and dimension. Figure 1 depicts screenshots of a visualization tool based on IDX being used to visualize a 530 MB IDX data set of a rat's retina scan. Figure 1(a) corresponds to visualizing data at the lowest resolution. This case requires querying a very small set of data. Figure 1(d) corresponds to visualizing data at higher resolution. This requires querying at multiple resolutions for a clipped viewing area. Figure 1(b) is the case where data visualized in (c) is zoomed with progressive increase in resolution whereas (c) corresponds to zooming without any progressive increase of detail, producing holes in images. Figure 1(e) and (f) are zoomed cross-sections from (b) and (c), and here the holes are clearly visible detectable.

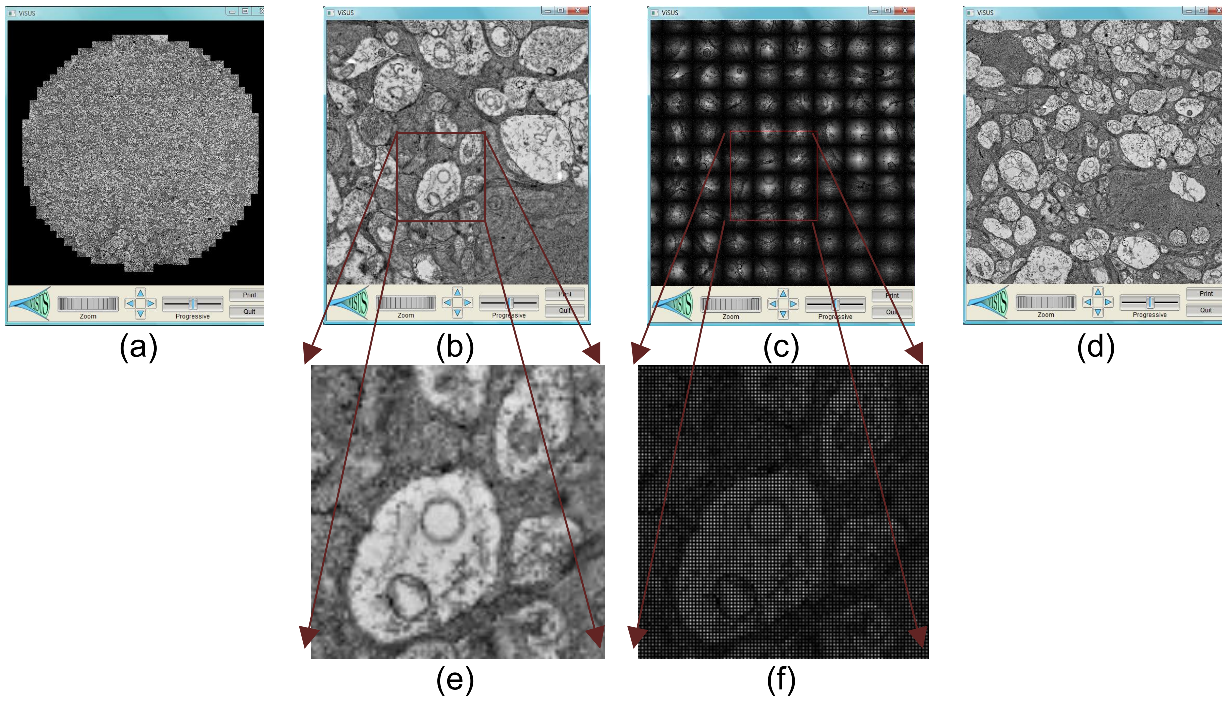

Figure 1. (a) lowest resolution data; (b) progressive zoomed data at high resolution (c) zoomed data at low resolution (d) data at high resolution (e) zoomed cross-section from b, hence high resolution (f) zoomed cross-section from c, hence low resolution with equally placed holes

Hierarchical Z (HZ) ordering is the key idea behind IDX data format. IDX supports multi-dimensional data of arbitrary dimensions and sizes. HZ order computation requires the spatial coordinates of data samples. For instance, it requires the x, y and z coordinates for a three dimensional data set. Exact formulation of HZ ordering can be found at [1]. Data is then reorganized into levels corresponding to the following formulation:

Level = floor ((log2 (HZ index))) + 1

These levels correspond to different resolutions the data is rearranged into. Level-wise data is then stored in IDX format data files. From file structure point of view, IDX file format has an .idx file that has all the required metadata (dimension, sample type, variable names and some more). The raw data is stored into a hierarchical level of binary files. The number of files and the size of the files are configurable.

TABLE I. TABLE SHOWING CONVERSION FROM X,Y,Z COORDINATES TO HZ COORDINATES

X	Y	Z	XYZ value	Z order	HZ order	Level
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	2	2	2	2
0	1	1	3	3	3	2
1	0	0	4	4	4	3
1	0	1	5	**5**	**6**	3
1	1	0	6	**6**	**5**	3
1	1	1	7	7	7	3

Table I demonstrates the conversion for a simple 2x2x2 volume of data. The conversion of data to IDX format can be considered as converting **n** dimensional data to one dimension. This conversion to HZ ordering is a bijective function, which is a required condition for parallelization. As a result of HZ ordering and corresponding distribution of data into different levels of resolution, it becomes increasingly fast to query just the required data set for analysis and visualization. For instance, there is little lag when zooming or panning a large scaled data at any desired rate. This is extremely critical for interactive visualization and analysis of data.

III. VISUS: A SERIAL IDX WRITER

The experiments in this paper were conducted on the Surveyor IBM Blue Gene/P (BG/P) system at the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory. Surveyor is a 4,096-core research and development system. Its storage subsystem consists of four file servers running PVFS and a DataDirect Networks S2A9550 SAN.

The first goal in our effort to utilize IDX in this HPC environment was to develop a parallel application that would use ViSUS I/O to write directly into IDX format. We developed a microbenchmark that divides an entire data volume into smaller 3D chunks, which each process independently writes to an IDX data set. MPI barriers and tokens are used to maintain order amongst processes; a process with rank **r** can write to an IDX file only after the process with rank **r–1** has finished writing. The processes cannot write concurrently due to conflicts in updating metadata and block layouts.

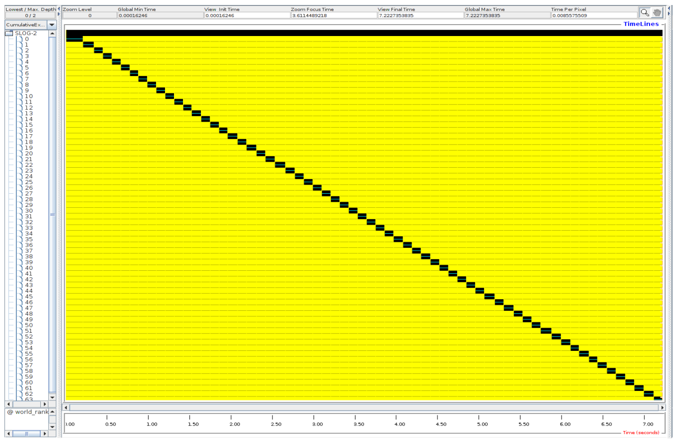

Figure 2. Jumpshot Image for the serialized ViSUS IDX writer using 64 nodes and a 64 MiB data set

We used MPE and Jumpshot [5] to understand the I/O patterns of the ViSUS microbenchmark,. Figure 2 depicts the Jumpshot profile of 64 processes writing an IDX file using ViSUS. Each horizontal line represents a process, where the yellow regions correspond to MPI barrier waits and black regions correspond to time spent writing data. As expected, we notice that a large portion of the runtime for each process is spent waiting for the I/O token.

The aggregate bandwidth for this benchmark on 64 processes as we increase the amount of data is illustrated in Figure 3. The efficiency improves as the aggregate volume to write is increased to 8 GiB, but its maximum performance is only 9.5 MiB/s. Using IOR, a widely adopted benchmark for parallel filesystems, we obtain a peak performance of 218MiB/s for 64 processes writing a total of 8GiB. Thus, we are able to achieve only 4% of the maximum throughput. This is expected as ViSUS is serial in nature and the various processes take turns to write the data out.

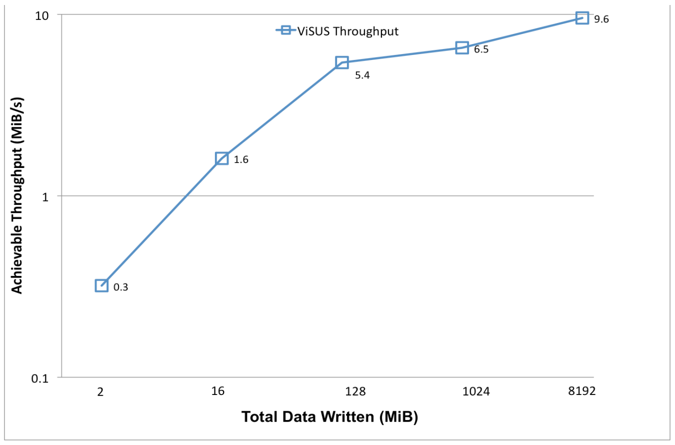

Figure 3. Performance of serialized ViSUS IDX writer using 64 processes as the volume of data is varied. The ViSUS IDX writer is able to achieve only 4% of the throughput achieved by IOR

IV. PIDX : Prototype API for Parallel IDX Writes

Based on our experience with the serialized ViSUS writer, we then developed a prototype API for performing concurrent I/O to an IDX data set. This API is called Parallel IDX (PIDX) and includes functions patterned after ViSUS for creating, opening, reading, and writing IDX data sets. Each of the PIDX functions is a collective operation that accepts an MPI communicator as an argument.

In both the ViSUS and PIDX API's, the dimensions and maximum volume of the data set is defined when the file is created. We therefore use the collective create function as an opportunity to pre-create all of the metadata file, subdirectories, and (initially empty) binary files that constitute the IDX data set. The rank 0 process is responsible for populating the metadata file and directory hierarchy. The work of creating the empty binary files is then distributed across all processes.

Once the data set is created, the PIDX write function can be used to collectively write arbitrary sub-volumes of data from each process. The data in this case is provided in the form of a contiguous, row-major ordered data buffer. Each process must calculate an HZ ordering for this sub-volume, reorder the data points accordingly, and write those data points to interleaved portions of the IDX data set. For prototype purposes, the PIDX library simply copies the sub-volume into an intermediate buffer when reordering. It also generates an index into that buffer indicating each level of the HZ hierarchy. Each level is then written in turn to the IDX data set using independent MPI-I/O write operations. Data within a single level is typically contiguous in file.

V. Optimization Strategies

The PIDX prototype described in the previous section greatly improved I/O performance over serial use of the ViSUS

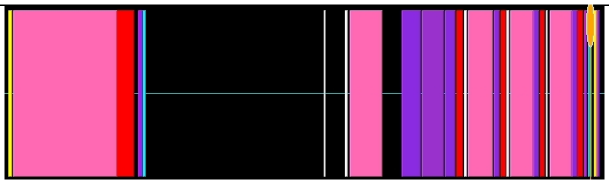

Figure 4. Jumpshot profile of a process writing IDX file without File Descriptor Caching. Pink depicts the file open time and the last three pink columns depict the redundant file opens being performed.

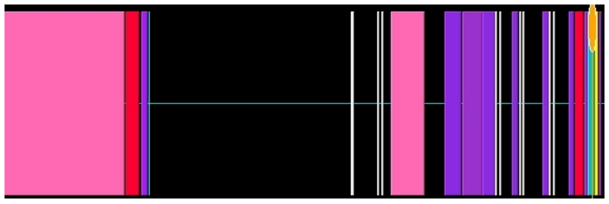

Figure 5. Jumpshot profile of a Process writing IDX file with File Descriptor Caching. The redundant file opens are eliminated via file descriptor caching

library. Jumpshot analysis revealed a number of inefficiencies, however. Figure 4 illustrates one example. This view highlights the time spent by rank 0 when writing data into the four initial HZ levels. The pink regions represent file open time. There is an initial expensive file open corresponding to creation of a binary file at PIDX create time, which cannot be avoided. However, it is also evident that a significant amount of time is spent in a sequence of four subsequent file open operations. This is because for each HZ level, PIDX identifies the appropriate binary file, opens it, writes a contiguous set of samples, and closes the file. However, the first two HZ levels only contain a single data point, the third level contains 2 data points, the fourth level contains 4 data points, and the number of data points doubles every successive level. In the initial HZ levels, the I/O cost was dominated by time spent opening the file.

In order to mitigate this overhead, we implemented a file handle caching mechanism in our prototype. When any process opens an underlying binary file, it holds the MPI file descriptor open for future use and does not close it until all I/O is complete. The result of this optimization is shown in Figure 5 for the same data set. There is now only one file open operation in the main write path, and performance is improved accordingly. Figure 6 depicts the performance improvement achieved with file handle caching over the default implementation for 64 cores as we increase the total data volume. We notice a significant improvement of up to 7-fold for data volumes of 128MiB. However, we notice only a marginal improvement with higher data volumes. This is because a significant amount of the I/O time was spent in the computation to generate the HZ ordering. We performed a detailed analysis of the HZ computation using the IBM BG/P universal performance counters and indentified bottlenecks associated with redundant computations as well as inadequate use of the floating point double hummers. By overcoming these, as depicted in Figure 6,we are able to achieve up to 75% improvement in I/O throughput over the file handle improvements and up to a 10-fold improvement over the default implementation.

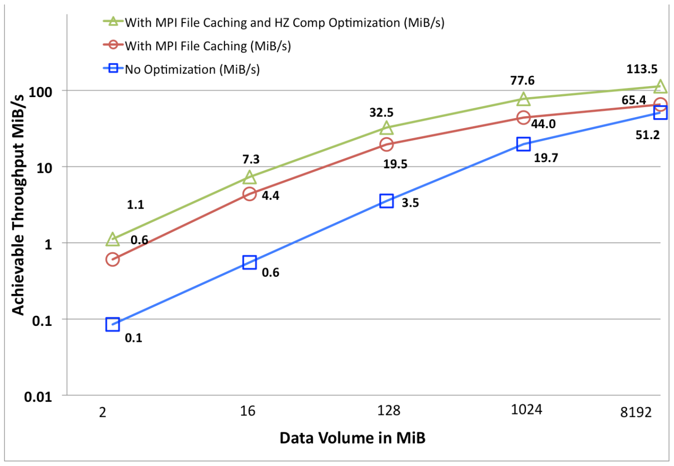

Figure 6. The achievable PIDX throughput on 64 cores as we vary the total data volume written with the various optimizations. File Caching and HZ Computation optimizations yield significant improvement in performance

Figure 7 shows the Jumpshot visualization of the PIDX microbenchmark when writing an IDX data set. The PIDX regions correspond to computation time, the pink regions correspond to file open time, and the purple regions correspond to I/O time. In contrast to Figure 2, which showed the serialized ViSUS writer, we are now able to efficiently utilize all processes when writing.

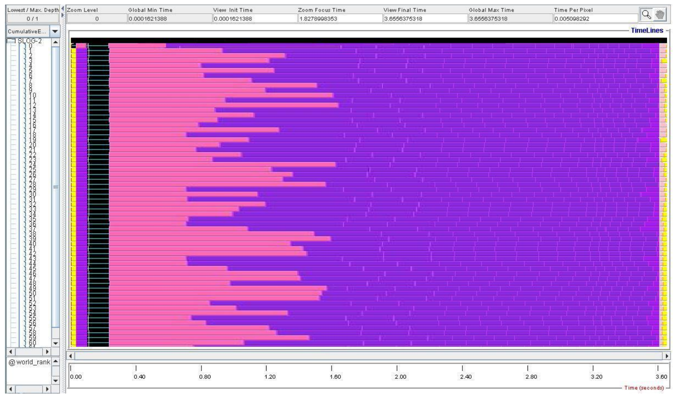

Figure 7. Jumpshot Image for the Parallel IDX (PIDX) writer using 64 nodes and a 16 MiB data set

VI. Performance Analysis of parallel IDX writer

In this section we evaluate the scalability of the PIDX prototype, both in terms of data volume and number of processes. We begin by investigating the weak scaling behavior of the PIDX library on Surveyor.

For weak scaling, we used a constant load of 128 MB per process. The total load was then gradually increased linearly as the number of processes was varied from 1 to 512, and the results are shown in Figure 8. From these results we see that a single process achieves approximately 6.85 MiB/s, comparable to the speed achieved by a serial writer for an equal volume of data. The peak aggregate performance of 406 MiB/s is reached with 512 processes. This is approximately 60% of the peak IOR throughput achievable (667MiB/s) on 512 cores of surveyor.

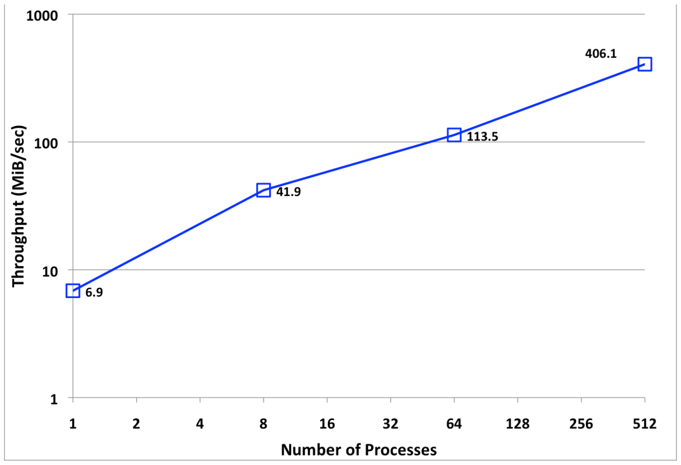

Figure 8. PIDX write throughput with weak scaling

Table II depicts the results of an experiment investigating strong scaling properties. In this case, the total data volume written was fixed at 8 GiB and we varied the number or processes writing from 8 to 512. The Surveyor compute nodes possess only 2 GiB of ram per 4 cores, so we were unable to evaluate smaller examples. As depicted in the figure the maximum performance is reached when using 512 processes.

TABLE II. PIDX STRONG SCALING FOR A TOTAL DATA VOLUME OF 8GIB

Number of Processes	PIDX Throughput in MiB/s
64	120.3
512	143.9

In both the weak and strong scaling examples, we found that we hit a limit on scalability with our current implementation, falling short of the peak surveyor write performance achieved by IOR. To investigate this issue further, we instrumented the time required to write each level of the HZ ordering. Table III depicts the time taken to write the various levels in the HZ hierarchy for a 8 GiB total volume on 64 nodes. A 8 GiB data volume consists of 30 HZ levels. We found that contention and metadata overhead caused levels 0 through 6 to take a disproportionate amount of time relative to the amount of data that they were writing. We plan to leverage aggregation strategies to better coordinate I/O in the first few cases where many processes contribute data to the same level of the IDX data set.

TABLE III. TIME TAKEN TO WRITE THE VARIOUS IDX LEVELS FOR A 8 GIB DATA VOLUME ON 64 PROCESSES

Level	Time (Sec)	Level	Time (Sec)	Level	Time (Sec)
0	**1.59693**	11	0.1291	22	0.3044
1	**1.4441**	12	0.1512	23	0.3630
2	**1.54467**	13	0.1518	24	1.9796
3	**1.5309**	14	0.1485	25	2.1405
4	**1.65302**	15	0.1029	26	2.7064
5	**1.64074**	16	0.085	27	4.605
6	**1.62843**	17	0.077	28	6.7606
7	0.1489	18	0.0869	29	10.533
8	0.13264	19	0.0876	30	18.251
9	0.13415	20	0.1406		
10	0.13281	21	0.2110		

VII. CONCLUSIONS AND FUTURE WORK

In this work we have shown that the IDX file format is a promising technology for analysis in scientific computing. We have also demonstrated that it is possible for simulation data to be written directly into IDX format via a parallel API that provides similar functionality to the ViSUS implementation but with an order of magnitude improvement in performance. We have also identified multiple optimizations that can be used to improve performance on the BG/P platform.

As noted in the previous section, we believe that further advances in PIDX efficiency can be achieved through the use of aggregation strategies that limit contention at the file system level. However, our current API is not ideal for this purpose. In future work we plan to revise the API in a manner that decouples the data model from the I/O mechanism. In particular, the application will describe the entire data set in its entirety (including support for multiple variables and discontiguous memory) up front before writing the data set. This will allow the PIDX library to take as much information as possible into account in order to schedule an efficient transfer mechanism. We will investigate multiple aggregation strategies using this platform.

ACKNOWLEDGMENT

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] Pascucci, V., and Frank, R.J., Hierarchical indexing for out-of-core access to multi-resolution data. Technical Report UCRL-JC-140581, Lawrence Livermore National Laboratory, 2001. A preliminary version was presented at the Lake Tahoe Workshop NSF/DOE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization.

[2] V. Pascucci and R.J. Frank. Global Static Indexing for Real-time Exploration of Very Large Regular Grids Conference on High Performance Networking and Computing archive proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)

[3] B. Summa, G. Scorzelli, M. Jiang, P.T. Bremer, V. Pascucci, Interactive Editing of Massive Imagery Made Simple: Turning Atlanta into Atlantis. ACM Transactions on Graphics - to appear, 2010

[4] V. Pascucci, D.E Laney, R.J. Frank, G. Scorzelli, L. Linsen, B. Hamann, And F. Gygi. 2003 Real-time monitoring of large scientific simulations. In ACM Symposiumon Applied Computing'03,ACMPress.

[5] W.Gropp, E.Lusk."User's Guide for MPE: Extensions for MPI Programs". 1998

[6] G. Bell, T. Hey, and A. Szalay. COMPUTER SCIENCE: Beyond the Data Deluge. Science, 323(5919):1297, 2009.

[7] Samuel Lang, Philip Carns, Robert Latham, Robert Ross, Kevin Harms, and William Allcock. I/O performance challenges at leadership scale. In Proceedings of Supercomputing, November 2009.